PRESBURGER ARITHMETIC WITH ARRAY SEGMENTS, PERMUTATION AND EQUALITY[1]

Louis E. Rosier

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

TR-83-17 September 1983

---

[1]This research was supported in part by the University Research Institute, the University of Texas at Austin and the IBM Corporation.

Table of Contents

ABSTRACT

In this paper we examine the validity (satisfiability) problem for a language containing Presburger arithmetic, fixed length array segments and some fixed second order predicates. In particular we consider the predicates PERM, which can be used to assert that two array segments contain the same multiset of elements and = (equality) which is used to assert that two array segments are identical (i.e. contain the same sequence of elements and have equivalent upper and lower array bounds). It is known that the aforementioned (unquantified) language has a decidable validity problem. (In fact, the satisfiability problem is NP-complete.) We consider the case when equality is defined in a seemingly weaker fashion. Let A $\simeq$ B be true if and only if array segments A and B contain the same sequence of elements. This definition of equality is more useful in the verification conditions of programs that manipulate queues and/or strings. However we show that when this form of equality replaces the earlier variety in the previously mentioned language, the validity problem becomes undecidable.

1. Introduction

Recently work has been done that augments the (logical) language of unquantified Presburger arithmetic with array segments, as well as some fixed second order predicates [5, 7-9, 13]. This work is motivated primarily by work in program verification and is concerned with deciding the validity (satisfiability) of such logical formulas. (The quantified theory is known to be undecidable [6].) The idea is to develop algorithms that will take as input annotated programs, complete with pre- and post-conditions and loop invariant assertions and check whether the program is correct. Since this problem, in general, is undecidable, recent research has focused on restricted classes of programs and assertions where the problem is decidable [5, 7-9, 13]. In this context, classes of programs that sort arrays have received a lot of attention [5, 7-9, 13].

In [13], formulas involving three data types were studied: boolean, integer and vector (of integer). Integer terms were constructed from the operations + and -, integer constants, integer variables and array access terms of the form X[i] where X is an array expression (a la [11]) and i is an integer expression. It was shown in [13] that the validity problem for unquantified formulas involving these terms, the relations = and $\leq$ (along with the usual logical connectives) was decidable. Restricted use of the predicate PERM(A,B) (which asserts that array expressions A and B define arrays which contain exactly the same elements, although perhaps not in the same sequence) in such formulas was also studied. The validity problem for the resulting class of formulas was also shown to be decidable. The satisfiability problem was, in fact, shown to be NP-complete.

In [7,9] (see also [8]) a decision procedure is described for a very restricted language that is capable of expressing the verification conditions for the ordering properties of certain sorting programs. In this language there were four data types: boolean, integer, array and an ordered type (the arrays contain only elements from the ordered type). Consequently array indices and array elements were not comparable. The formulas also involved array segments X[i..j], where X is an array variable and i and j are integer variables. Some restricted sorting programs can be verified using these techniques. Related results also appear in [3].

In [5] formulas involving five data types were considered: boolean, integer, an ordered type, finite multisets (multisets with possibly negative multiplicities were actually considered in [5], however the usual multiset was represented in the language hence we ignore the extension), and finite length arrays (both multisets and arrays were over the ordered type). Many predicates and/or operators involving the various data types were allowed in this language including predicates to assert that an array is ordered and that the multisets represented by two distinct arrays were the same, as well as the usual integer operations and relations. Consequently the decision procedure given for this language yields a technique that can be used to verify many sorting problems, as well as programs concerning other data structures (e.g. stacks). The satisfiability problem was again shown to be NP-complete [4]. Although the aforementioned languages are not always comparable the most comprehensive language is the latter one.

In [12] an interesting language or notation was described that is suitable for reasoning about arrays. Although decision procedures were not of interest in [12], many useful concepts were illustrated with respect to reasoning about arrays. Thus many predicates about arrays and their contents were introduced. Not surprisingly then is the fact that there is a great deal of overlap between the predicates studied in [12] and those mechanized in [5]. As a comparison vehicle this work can be used to point out some of the restrictions of the language in [5] and suggest additional types and/or predicates that perhaps can also be mechanized along with those which already are. (In fact two such extensions are suggested in [5].) The first such extension is that of "sets." While the language in [5] can reason about the multiset of elements in an array it appears unable to reason about the set of elements in an array. It does not seem unreasonable to believe that finite sets (along with some appropriate operators and predicates) could be added to this language and that a suitable decision procedure found to decide the validity of such formulas. This seems plausible when one considers other decidable theories that are in some sense able to reason about sets [1]. The second extension concerns the vector concatenation operator or the shift equivalence relation. In [5] two arrays are equal if and only if their contents (in order) are equal and they have the same lower and upper bounds. This constrains the formulas in such a way that most programs that use queues and/or strings cannot be verified (using those techniques anyway). In order to extend this work to such programs it seems necessary to require a weaker predicate for array equality--one that merely asserts that the respective sequences of elements are the same.

In this paper we consider the latter problem. We start with a restricted form of the language in [5] and add the predicate that asserts that two arrays are "equal as sequences" (but asserts nothing about the respective array bounds), and show that the validity problem is undecidable for the resulting language. In our proof we show a somewhat more restricted result which does not need many of the operators and predicates available in the language. The proof illustrates that integer division can be encoded in such formulas and hence the undecidability of the validity problem for this language follows from the well known results in [10].

2. The Logical Assertion Language

The language we consider has four types: boolean, integer, an ordered type and array (over the ordered type). The specifics of the ordered type are not important except that there must be at least two distinct elements. The logical operators are the usual $\neg$ (not), $\wedge$ (and) and $\vee$ (or). The integer operators are +, -, < and =. The formal language includes the integer constant symbols ...,-2,-1,0,1,2,..., integer variables (which we will denote as i, j, k etc. and array variables (denoted by A, B,...). An array segment is of the form A[i..j] (i(j) is called the lower (upper) bound of the segment), where A is an array variable and i and j are integer terms (defined from integer constants and integer variables in the usual manner). In addition we consider the following predicates on array segments. Let M and N denote array segments.

1. PERM(M,N)

2. M = N

3. ORD(M)

The predicate PERM(M,N) is true if and only if the two array segments M and N contain the same multiset of elements (and thus are the same length). The predicate M=N is true if and only if the two array segments M and N contain identical sequences of elements and have identical lower and upper array bounds. The predicate ORD(M) is true if and only if the elements in the array segment M appear in order (as defined by the ordered type).

The aforementioned language is a subset of the much richer language described in [5] (i.e. each formula in this language is expressible in the language of [5]). No attempt has been made here to illustrate all the operators, types, and/or predicates that are considered in [5]. (In fact we only consider a very small number of them.) However the ones mentioned here seem crucial in terms of asserting the important properties about programs over these types of variables. For instance the introduction of multisets in [5] was basically done in order to express the PERM predicate. The equality and order predicates are important in that they seem necessary to reason about array segments in the types of verification conditions that arise in sorting programs.

To this list of predicates we add a single predicate. Again, let M and N denote array segments.

1. $M \simeq N$

The predicate $M \simeq N$ is true if and only if the two array segments M and N contain identical sequences of elements (and thus are the same length). Note that M=N if and only if $M \simeq N$ and the lower and upper array bounds of M are equal to the lower and upper array bounds of N. Denote the aforementioned language by L.

3. The Undecidability Result

In this section we show that the language L summarized in the previous section has an undecidable satisfiability (validity) problem. The proof uses the $\simeq$ predicate to force an array segment to be composed of repeated patterns of elements. Then other predicates are used to force a variable to express the integer division of two integer terms.

Theorem. The language L has an undecidable validity (satisfiability) problem.

Proof. Since any formula of Presburger arithmetic can essentially be expressed in L, it is well known that if in addition the "/" operator (integer division) were expressed that the problem in question would be undecidable [2]. (This follows from the undecidability of Hilbert's tenth problem [10], the problem of deciding for any given polynomial with integer coefficients whether it has a nonnegative integral solution.) Consequently it is sufficient to show that the addition of the $\simeq$ predicate allows integer division to be expressible. Consider the following clauses definable in L:

$C_0 \equiv (1 < j \wedge j+1 < w \wedge 4 \leq i)$

$C_1 \equiv PERM(A[1..w], B[1..w])$

$C_2 \equiv A[1..i-2] \simeq A[2..i-1]$

$C_3 \equiv B[1..j-1] \simeq B[2..j]$

$C_4 \equiv B[j+1..w-1] \simeq B[j+2..w]$

$C_5 \equiv B[w] \simeq A[1]$

$C_6 \equiv B[1] \simeq A[i]$

$C_7 \equiv \neg(B[1] \simeq B[w])$

$C_8 \equiv A[1..i] \simeq A[1+i..w+i]$

Now consider what happens when $C = \bigwedge_{n=0}^{8} C_i$ is true. For C_1 to be true, A[1..w] and B[1..w] must contain exactly the same multiset of elements. C_2 is true if and only if A[1]=A[2]...=A[i-1]. C_3 is true if and only if B[1]=B[2]=...=B[j] and C_4 is true if and only if B[j+1]=B[j+2]=...=B[w]. Thus clauses C_1-C_7 imply that array segments A[1..w] and B[1..w] each contain only two distinct elements (the elements B[1] and B[w]). Furthermore the integer term j must contain exactly the number of times the element B[1] (or equivalently A[i]) appears in the array segment A[1..w], since by C_3 and C_4 the array segment B[1..w] has all occurrences of the first element preceding all occurrences of the second element. Now C_8 is the important clause and shows the real power of the $\simeq$ predicate. For C_8 to be true it must be the case that the sequence of elements in A[1..w+i] is periodic with a period of i. Thus the sequence of elements in A[1..w] is the sequence of elements in A[1..i] repeated w/j times (with perhaps another partial repetition left over). Consequently since the number of occurrences of this element A[i] appearing in A[1..i] is one, we have that the number of occurrences of A[i] appearing in A[1..w] is w/i. Hence whenever C is true j=w/i. Since w/i can be expressed by a Presburger formula whenever C_0 is false we are finished. The theorem now follows from [10]. []

The reader should note that the above proof only used predicates PERM and $\simeq$. Although a shorter proof of this result can be shown using other predicates available in the language presented in [5], this proof serves to show the limitation of even a restricted form of such a language when the weaker form of equality is expressible. Thus reasoning about arrays with the weak form of equality is much more difficult than before. Also it seems difficult to imagine a weaker language containing the predicate $\simeq$, that is capable of making interesting and useful assertions. Since the $\simeq$ predicate seems to be necessary for reasoning about most programs that manipulate queues and/or strings, any decidable language considered for this purpose will not be able to express the permutation predicate. (However some programs that manipulate queues do so by adding to the right (removing from the left) of an array, but never shift the array contents. These can be handled without the $\simeq$ predicate.)

Acknowledgement

I would like to thank David Jefferson for helpful discussions.

References

1. Buchi, J., On a decision method in restricted second order arithmetic, Proc. Internat. Cong. Logic, Methodology and Philos. Sci. 1960 (E. Nagel, P. Suppes, and A. Tarski, eds.), pp. 1-11, Stanford Univ. Press, Stanford, California, 1962.

2. Enderton, H., A Mathematical Introduction to Logic, Academic Press, NY, NY, 1972.

3. Jaffar, J., Presburger arithmetic with array segments, Information Processing Letters, Vol. 12, No. 2, April 1981, pp. 79-82.

4. Jefferson, D., Personal communication, 1983.

5. Jefferson, D., Type reduction and program verification, Ph.D. Dissertation, Carnegie-Mellon University, April 1980.

6. Lifshits, V., Some reduction classes and undecidable theories, in "Studies in Constructive Mathematics and Mathematical Logic," Pt. I, A. Slisenko, Ed., "Seminars in Mathematics," Vol. 4, V. A. Steklov Mathematical Institute, Leningrad, 1969, pp. 24-25.

7. Mateti, P., A decision procedure for a class of sorting programs, Technical Report 78/1, Dept. of Computer Science, University of Melbourne, Australia (1978).

8. Mateti, P., A decision procedure for the correctness of a class of programs, J. ACM, Vol. 28, No. 2, April 1981, pp. 215-232.

9. Mateti, P., An automatic verifier for a class of sorting programs, Ph.D. Thesis, Dept. of Computer Science, University of Illinois, Urbana, IL, 1976, Tech. Rep. UIUCDCS-R-76-832.

10. Matijasevic, Y., Enumerable sets are Diophantine, Dodl. Akad. Nauk. SSSR 191 (1970), pp. 279-282.

11. McCarthy, J., Towards a mathematical science of computation, Proc. IFIP Congress 1962 (North-Holland, Amsterdam, 1972), pp. 21-28.

12. Reynolds, J., Reasoning about arrays, Commun. ACM, Vol. 22, No. 5, May 1979, pp. 290-299.

13. Suzuki, N. and Jefferson, D., Verification decidability of Presburger array programs, J. ACM, Vol. 27, No. 1, January 1980, pp. 191-205.